Tuning DeSR for the Evalita 2011 Dependency Parsing

Giuseppe Attardi, Maria Simi, Andrea Zanelli

Università di Pisa, Dipartimento di Informatica, Largo B. Pontecorvo 3,
56127 Pisa, Italy
{attardi, simi, andreaz}@di.unipi.it

Abstract. DeSR is a statistical transition-based dependency parser that learns from a training corpus which action to take to build the parse tree while scanning a sentence. DeSR can be configured to use different feature models and classifier types. We tuned the parser for the Evalita 2011 corpora by performing several experiments of feature selection and also by adding some new features. The submitted run used DeSR with two additional techniques: (1) *reverse revision parsing*, which addresses the problem of long distance dependencies, by extracting hints from the output of a first parser as input to a second parser running in the opposite direction; (2) parser combination, which consists in combining the outputs of different configurations of the parser. The submission achieved second best accuracy. An analysis of the errors shows that the accuracy is quite high on half of the test set and lower on the second half which belongs to a different domain.

Keywords: Dependency parser, shift-reduce parser, stacked parser, parser combination.

1 Description of the System

DeSR (Dependency Shift Reduce) is a transition-based statistical parser [7] [9] which builds dependency trees while scanning a sentence and applying at each step a proper parsing action selected through a classifier based on a set of representative features of the current parse state [1]. Parsing is performed bottom-up in a *Shift*/*Reduce* style [1], except that the parsing rules are special and allow parsing to be performed deterministically in a single pass [2].

The state of the parser is represented by a triple $\langle S, I, A\rangle$, where I is a sequence of tokens still remaining in the input. Initially I contains the sequence of tokens for the sentence being parsed: each token contains the word w_i as well as a set of word features p_i, consisting typically of the POS tag, the word lemma and possibly morphological features. S is the stack containing analyzed tokens; A is a set of labeled dependencies constructed so far. At each step, the parser selects a parsing rule to apply and modifies its state accordingly.

It is possible to specify, through a configuration file, the set of features to use (e.g. POS tag, word lemma, morphological features) and the classification algorithm (e.g.

Multi-Layer Perceptron, Support Vector Machine, Maximum Entropy). The parser can use beam search in conjunction with those classifiers that provide a probability distribution for the predictions, i.e. Maximum Entropy and Perceptron. Moreover the parser can be configured to run either in left-to-right or right-to-left word order.

A quite effective use of DeSR is the *Reverse Revision* parser [4], a *stacked parser* [3] which first runs in one direction, and then extracts hints from its output to feed another parser running in the opposite direction. A Reverse Revision parser was used successfully in several parser competitions, including Evalita 2009 [5] and Icon 2010 [6].

All these options allow creating a number of different parser variants, all based on the same basic algorithm. This allows selecting the most effective variants and then a further improvement can be achieved by the technique of parser combination [4]. For parser combination we use a greedy algorithm, which hence preserves the linear complexity of the individual parsers and often outperforms other more complex algorithms [8]. In the Evalita 2009 experiments, the algorithm was able to reduce the error rate up to 8% in the pilot task on dependency parsing.

For the Evalita 2011 competition, we used a configuration similar to that for Evalita 2009 and performed a number of experiments to improve feature selection.

2 Experiments

We merged the six corpora of TUT in a single *initial corpus*. We produced from the *initial corpus* a new corpus by rewriting the morphological information in the FEATS field in a standard format and accommodating other information in the same field in the fine-grained PoS column and two new columns: one column EXTRA that contains additional morphological information and one column SEM with semantic information. The format file of DeSR was modified to include these two extra columns in order to exploit this additional information. We will call *base corpus* the corpus with the two new columns.

During development the *base corpus* was divided randomly into: a training set (93% of sentences) for model training and a development set (7%) for model testing.

Starting from the configurations that gave the best results in the Evalita 2009 Dependency Parsing main task, we performed a feature selection process by adding and deleting individual features, and verifying the improvements brought by each of them. For each set of features we tried as classifiers both Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM). Moreover, for each set of features, both the normal parser and the stacked parser were run, in both directions.

The feature selection process produced about 170 different models and as many parse results. This was possible since the parser is fast enough that training can take about 3 min. in a typical configuration on a Linux server with an Intel® Xeon® 2.53GHz CPU. The 25 best configurations were tested in combinations of 3 or 4, using the method described in [2]. The configurations of the four best parsers were chosen for the final run by training four parsers on the whole *base corpus* and combining their output.

The four parsers used for the final run share a set of common features as reported in Table 1.

Table 1. Common features of all parsers.

Feature	Value
LEMMA	-2 -1 0 1 2 3 prev(0) leftChild(-1) leftChild(0) rightChild(-1) rightChild(0)
POSTAG	-2 -1 0 1 2 3 next(-1) leftChild(-1) leftChild(0) rightChild(-1) rightChild(0)
CPOSTAG	-1 0 1
FEATS	-1 0 1
DEPREL	leftChild(-1) leftChild(0) rightChild(-1)
LexChildNonWord	true
StackSize	true
VerbCount	true
PastActions	1

All four parsers are stacked parsers, which use an additional set of common features for the second stage, as reported in Table 2.

Table 2. Common features for reverse revision parsers.

Feature	Value
PHLEMMA	-1 0 1
PDEP	-1 0 1
PHPOS	-1 0 1

The specific features used by the four parsers used for the final run are listed in Table 3. Most differences lie in the fields EXTRA and SEM is because all the best parsers found with the feature selection process differed mainly in such fields, in fact, variations on the other features (with respect to the common configuration reported previously) led to a decay in performance.

A further experiment was performed to assess the utility of features expressing morphological agreement between words, either in gender and number. The best configurations were tested by adding the feature `MorphoAgreement`. We considered two different ways to represent morphological agreement:

- Adding features `=N` or `=G` to express the cases when the top and next token agree in number or gender respectively (neutral agrees with any other value)
- Adding features `!=N` or `!=G` to express the cases when the top and next token do not agree

The second alternative avoids that a missing feature would be considered as a disagreement. For example in “potuto essere vista”, “potuto” and “vista” are indirectly connected even though there is no gender agreement. In both cases also `=NG!1` and `=NG!2` is added if the top token disagrees either in gender or number with the second or third token on the input respectively.

While the addition of `MorphoAgreement` features corrects some errors due to wrong agreement, it introduces errors in other cases. Overall the accuracy improves slightly in about half of the runs (with an average variation of 0.3%) but is slightly worse in the others, and hence the effectiveness of the feature remains questionable. The best score on the development set (LAS 87.78%) was achieved by the combination among parsers without `MorphoAgreement`, with a small margin with respect to the combination of parsers with the feature (LAS 87.70%).

Table 3. Comparison between the four best parsers.

Parser	Type	Classifier	Features	Stacked Parser Features
1	Forward Revision	MLP	EXTRA -1 0 SEM -1 0 1 LexCutoff 0	EXTRA -1 0 SEM -1 0 1 PLOC -1 0 1 LexCutoff 0
2	Reverse Revision	MLP	EXTRA -1 0 SEM -2 -1 0 1 2 3 LexCutoff 2	EXTRA -1 0 SEM -1 0 1 PLOC -1 0 1 LexCutoff 0
3	Reverse Revision	SVM	EXTRA -1 0 SEM -2 -1 0 1 2 3 LexCutoff 2	EXTRA -1 0 SEM -1 0 1 PLOC -1 0 1 LexCutoff 0
4	Forward Revision	SVM	EXTRA -1 0 1 SEM -2 -1 0 1 2 3 LexCutoff 2	EXTRA -1 0 1 SEM -2 -1 0 1 2 PLOC 0 1 PHDEP -1 0 1 PHHLEMMA 0 1 LexCutoff 2

3 Results

Table 4 reports the values of Labeled Attachment Score (LAS) and Unlabeled Attachment Score (UAS) achieved by the four individual parsers and by their combination on the development set.

Table 4. Results of the four parser and their combination on the development set.

Parser	**LAS on Dev Set**	**UAS on Dev Set**
1	85.34 %	89.49 %
2	86.67 %	90.55 %
3	85.90 %	89.89 %
4	85.05 %	88.92 %
Combination	87.78 %	91.40 %

We used this parser combination for our official submission which achieved the official scores reported in Table 5, compared with the scores of the best submission for this task.

Table 5. Final result on the Evalita 2011 test set.

Run	LAS	UAS
EVALITA_11_PAR_DEP_UNIPI	89.88	93.73
Best submission to Evalita 2011	91.23	96.16

4 Discussion and Error Analysis

The official test set consists of 150 sentences from the Civil Law domain and 150 sentences from other domains. The parser achieves excellent accuracy on the first portion of the test set (92.85%), while the score drops significantly on the rest of the test set (86.61%), as shown in the following table.

Table 6. Breakdown of accuracy on the test set.

Test Set	LAS	UAS
tut_test	89.88	93.73
tut_test_law	92.85	96.18
tut_test_rest	86.61	91.04

A detailed analysis shows the following distribution of head errors according to the CPOS tag of the token, in the two subparts of the test set (Civil Law and Rest).

Table 7. Breakdown of errors according to head CPOS

CPOS	Civil Law		Rest	
	Head Errors	%	Head Errors	%
NOUN	5	1	20	2
PREP	48	8	67	12
ART	8	1	19	4
VERB	17	2	45	10
PUNCT	31	10	109	27
ADJ	6	3	8	3
PRON	3	1	5	3
ADV	4	3	15	11
CONJ	26	13	134	19

There is a considerable increase in errors for punctuation and conjunctions. Simply discarding the punctuation errors would increase the accuracy to 91.6 %. An analysis of the errors on punctuations led to grouping them in the following categories:

Top error	errors due to incorrect identification of parse tree root
Parenthetical	error in commas surrounding a parenthetical phrase
Apposition	error in commas separating an apposition
Coordination	errors in coordinate attachment
Balance	errors in balancing punctuations, quotes or parentheses.

Indeed the parser has often difficulty in deciding where to attach a comma, since when the comma is reached it has constructed the trees for phrases preceding the comma, but it only can see individual tokens after the comma.

For example, in the sentence "… draft, cioè una bozza …", it would need to figure out that "bozza" relates to "draft". However "bozza" is a child of "una", which is a child of "cioè", which is a child of comma, which is a child of "bozza". Hence, in order to figure out that the comma is a way to relate "draft" with "bozza" it would have to look 4 token forward and be sure that the intervening tokens do not relate to something else. In order to handle this problem we have experimented with a variant of the parsing algorithm that delays *Left* reductions until the phrases on the right have been parsed. This requires also introducing an *UnShift* operation, in order to resume the *Left* reduction at the proper time. Unfortunately some early experiments with this algorithm showed negative effect on accuracy.

References

1. Aho, V., Ullman, J.D.: The Theory of Parsing, Translation and Compiling, vol. 1. Prentice-Hall, Englewood Cliffs, NJ (1972)
2. Attardi, G.: Experiments with a Multilanguage non-projective dependency parser. In: Proceedings of the Tenth CoNLL (2006)
3. Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., Yuret, D.: The CoNLL 2007 shared task on dependency parsing. In: Proceedings of the CoNLL 2007 Shared Task. Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL) (2007)
4. Attardi, G., Dell'Orletta, F.: Reverse Revision and Linear Tree Combination for Dependency Parsing. In: Proceedings of NAACL HLT (2009)
5. Attardi, G., Dell'Orletta, F., Simi, M., Turian, J.: Accurate Dependency Parsing with a Stacked Multilayer Perceptron. In: Proceedings of Workshop Evalita 2009, ISBN 978-88-903581-1-1 (2009)
6. Attardi, G., Dei Rossi, S., Simi, M.: Dependency Parsing of Indian Languages with DeSR. In: Proceedings of ICON-2010 tools contest on Indian language dependency parsing, Kharagpur, India (2010)
7. Nivre, J., Scholz, M.: Deterministic Dependency Parsing of English Text. In: Proceedings of COLING 2004, pp. 64–70, Geneva, Switzerland (2004)
8. Surdeanu, M., Manning, C. D.: Ensemble Models for Dependency Parsing: Cheap and Good? In: Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (NAACL-2010) (2010)
9. Yamada, H., Matsumoto, Y.: Statistical Dependency Analysis with Support Vector Machines. In: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT), pp. 195–206 (2003)